A Practical Experience of How to Teach Model-Driven Development to Manual Programming Students

José Ignacio Panach[*,a], Óscar Pastor[b]

[a] Escola Tècnica Superior d'Enginyeria, Departament d'Informàtica, Universitat de València, Avinguda de la Universitat, s/n 46100 Burjassot, Valencia, Spain
[b] Valencian Institute of Research in Artificial Intelligence (VRAIN),Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain

Abstract. *This paper presents the teaching experience of a course named Information Systems Engineering in a Master's degree program of the Universitat Politècnica de València. The target of this course is to teach Model-Driven Development (MDD). On the last years we have observed that students attended the course with poor motivation since they do not see MDD as being a useful development paradigm. The students have an extensive background in a traditional method (they are good programmers) where all the code is manually programmed, but they lack sound experience in conceptual modeling. In order to improve their motivation and to highlight the pros and cons of MDD, we propose a practical comparison of a traditional method and MDD. The teaching methodology consists of a problem-based learning task where students must develop two problems from scratch, one with a traditional method and the other with MDD. Our experience has been evaluated in terms of attitude towards MDD, knowledge of MDD, quality of the developed system, and satisfaction of the developer. The results show that the students obtained significantly better results for MDD in terms of attitude, knowledge, and quality.*

Keywords. Model-Driven Development • Conceptual Modeling • Code Generation

Communicated by Kristina Rosenthal, Estefanía Serral, Monique Snoeck, Stefan Strecker. Received 2021-05-06. Accepted after 3 revisions on 2022-12-23.

1 Introduction

Model-Driven Development (MDD) (Atkinson and Kühne 2003) is a software development paradigm that aims to raise abstraction levels to specify all system features. In the same way as a Java compiler takes a textual implementation as input and generates machine instructions, MDD aims to specify the system through models that are the input for a model compiler that generates the code. This way the developer does not focus all of the effort on implementation issues, but on models that abstractly represent the system. Model-to-code transformations can be done through transformation rules. The underlying motivation for MDD is to improve productivity (Selic 2003), quality (Singh and Sood 2009), effort (Hailpern and Tarr 2006), and satisfaction (Martínez et al. 2013).

The MDD paradigm is based on the concept of Conceptual-Model Programming (Embley et al. 2011), which states that programming activities can be carried out via conceptual modeling. Conceptual-Model Programming proposes that developers focus on the domain problem through conceptual models, relegating the solution space (specific code) to model-to-code transformations.

* Corresponding author.
E-mail. joigpana@uv.es
This work was developed with the support of the Spanish Ministry of Science and Innovation project SREC (PID2021-123824OB-I00) and was co-financed by ERDF. It also has the support of Generalitat Valenciana with GISPRO project (PROMETEO/2018/176) and GENI (CIAICO/2022/229).

Thus, the combination of MDD and Conceptual-Model Programming leads to software development based on models (Pastor and Molina 2007).

In general, the syllabi of most degree programs in Computer Science focus on courses to teach the skill of coding. For example, in the Degree in Computer Science at Universitat Politècnica de València (UPV), 10% of the courses are about industrial management, 16% are about mathematics and physics, 16% are about network and hardware and 58% are about software development (operating systems, programming, artificial intelligence, algorithms, etc.). However, there is not any course for conceptual modeling under the context of MDD. There is only one course of software engineering where students learn about UML diagrams. However, these diagrams are used as a way to document the system; in the end, students have to manually implement the code of the system. These numbers indicate that we are training developers for manual programming jobs. Students that finish the degree program are good programmers, but they do not consider models as being a useful tool. These models are conceived only for documenting, and their practical use is very limited. Moreover, students feel that working with these models is boring and a waste of time since models become easily obsolete as soon as the code evolves.

The first course that deals with MDD at Universitat Politècnica de València (UPV) appears in the "Master of Engineering and Software Systems Technology" program. The name of the course is "Information Systems Engineering", with 3 ECTS credits. Both of the authors of this paper are teachers of this course. The main challenge of the course is to motivate the students to work with models instead of coding. Note that previous teachers have been teaching the development of systems through coding for at least the last four years. Therefore, it is difficult for students to build models to abstractly represent the system. Moreover, this course is the first time that we have introduced the concept of MDD and Conceptual-Model Programming to the students. In general, the first reaction of the students is to think that all of the theoretical benefits that the literature states about MDD are fake, and they think that they are better at working with a traditional software development method. By a traditional method, we mean a method where the developer must manually write the code that implements the system. The use of models, in this case, is only for documentation or as instruments to report solutions, with no option to generate code from them.

To make students aware of their own actions, we organized the course as a comparison between MDD and a traditional software development method. The main contribution of the article is the description of this practical experience. Each student compares her/his quality and satisfaction with software development through a traditional method versus development that is based on MDD. This way, the students can draw their own conclusions about MDD use and the pros and cons. Since our students have a very good background in working with a traditional method, the course focuses only on teaching how to work with the MDD paradigm using the WebRatio tool (WebRatio 2021). The course is based on a Problem-Based Learning methodology where students must develop the solution for two problems from scratch: one using a traditional development method and the other using MDD. Both problems are different in order to prevent the learning effect, but they are similar in difficulty. Therefore, the students can compare both software development methods by working with each of them.

The results of our teaching experience are analyzed using four variables. First, we measured the students' **Attitude** towards MDD and their **Knowledge** of MDD. Both variables were evaluated at the beginning of the course and at the end of the course using the same questionnaires. This way we were able to compare whether or not there were significant differences between the answers at both moments. We also measured the **Quality** of the developed system and the developers' **Satisfaction** for each problem.

The results show significant differences for all of the variables. Attitude and Knowledge show better results after applying our teaching methodology. Quality shows better results for the MDD development. For Satisfaction, the students consider MDD as being easy to use, but they have no intention of using MDD in the future. In our analysis, these numerical results are complemented with assumptions about the pros and cons of MDD, which were obtained through open questions in the discussion session of the course.

The paper is structured as follows. Section 2 describes related works with practical experiences in teaching MDD in courses. Section 3 explains the teaching methodology that we used in our experiment. Section 4 presents the design of the validation of our experience. Section 5 presents the results of the evaluation. Section 6 discusses the results and explains them. Finally, Section 7 presents conclusions and future work.

2 Related Works

There are several previous works that have conducted an analysis of trends in MDD and conceptual modelings, such as Härer and Fill (Härer and Fill 2020) and Chen et al. (Chen et al. 2007). From all the previous works in MDD and conceptual modeling, this related works section focuses on those that deal with the pedagogical point of view. There are different options to conduct a literature review (Watson and Webster 2020), and we opt for a Targeted Literature Review (TLR). TLR is a non-systematic, in-depth, and informative literature review aimed at keeping only the significant references, maximizing rigorousness while minimizing selection bias (Huelin et al. 2015). As TLR protocol to look for papers, we have structured this search in terms of the search string, inclusion and exclusion criteria, and search procedure. Since the target domain of study is "teaching MDD", all synonyms and related concepts must be considered in our search. These include: Model-Driven Development, Model-Driven Engineering, Model-Driven Architecture, and Conceptual modeling. All of these terms led to the following search string: ("Model-Driven" OR "Conceptual modeling") AND ("Teaching" or "Teach"). This search string was applied to the title, keyword, and abstract in February 2021 in Scopus, IEEE Xplore, and ACM Digital Library. Apart from the search string, we used two publication outlets: the workshops of the ER (Conceptual modeling) conference proceedings under the name of *Advances in Conceptual modeling* and the *ICSE* conference. ICSE is the most relevant conference of Software Engineering and has special tracks for teaching education. Advances in Conceptual modeling has several special workshops related to teaching conceptual modeling. The exclusion criteria include: (1) tutorial papers; (2) papers that do not report the results of the teaching experience; and (3) papers without models. The inclusion criteria include:(1) papers that describe the teaching methodology in MDD through conceptual modeling; and (2) papers that describe how they evaluated the teaching methodology. The publications outlets yielded three papers from Advances in Conceptual Modeling and two papers from ICSE. The search string returned 226 papers. After applying the exclusion and inclusion criteria to the title and the abstract and gathering papers from both the outlets and the search string, we analyzed the content of 18 papers, which we describe below. The dataset with the analyzed papers is presented in (Panach and Pastor 2021a). The entire search process was conducted by one of the authors of the current paper.

There are several works in the literature that deal with how to teach MDD to students. The work of Rosenthal et al. (Rosenthal et al. 2019) contributes with a systematic literature review with 121 publications to look for the importance of teaching and learning conceptual modeling and its accepted challenges. The proof of this variety is the fact that there are works that analyze and compare several existing teaching techniques in the area of MDD, such as the work developed by Ciccozzi et al. (Ciccozzi et al. 2018). Those authors conducted a survey of 47 MDD instructors to analyze tools, technologies used, and positive and negative factors affecting learning outcomes. The results show the prevalence of assessment methods that focus on tools and technologies. There are works that focus on the teaching methodology at the requirements elicitation stage. For example, the work of Reyes and Quintero (Reyes and Quintero 2020) teaches how to elicit requirements in an MDD environment based on scenarios with real problems. The students had to build Use Case Diagrams to elicit requirements and transform those models into code manually. The classes were virtual and based on active learning principles in small groups. The results indicate sufficient motivation, greater retention of the learned abilities, and improvement in interpersonal relations. The work of Paja et al. (Paja et al. 2015) reports the experience in teaching conceptual modeling with the i* goal-oriented language. The results demonstrate that i* allows students to evaluate the satisfaction of goals in their models and to better understand their models. There are no transformations among models in this study. There are works such as Berre et al. (Berre et al. 2018) that aim to generate code from the conceptual models. That work describes the experience of teaching several models from requirements such as BPMN, Canvas, and IFML to generate code through MDD tools (e. g., WebRatio). The results show that the use of executable models at an early stage in the course is key in training students. Other works, such as Zibri et al. (Zribi et al. 2016), focus on the design of a framework to simulate models. Zribi et al. defined a framework to simulate BPMN models that include event-based monitoring, allowing collaborative simulation and providing learners' assessment. As a conclusion of our review of all of these related works that deal with requirements models, we highlight that only one of them (Berre et al. 2018) considers code generation in the course. Most of the existing works focus on evaluating models instead of code, which could hide the utility of the method from the point of view of the students.

Other proposals are based on the concept of gamification, such as Larenas et al. (Larenas et al. 2018). Those authors used a role-playing game named Classutopia to teach the design of a class diagram from the point of view of conceptual modeling. The game aims to teach about modeling challenges and comprehension of models with different complexity levels. The work of Roungas and Dalpiaz (Roungas and Dalpiaz 2016) proposes an MDD framework to design serious games for teaching environments. That work defines the primitives required to define a game for teaching a concept of MDD. It is important to note that existing works in gamification focus on models; there is no work that studies code generation.

There are proposals that teach MDD from an agile development point of view. The work of Ghiran et al. (Ghiran et al. 2020) describes an experience of teaching conceptual modeling through a metamodeling approach. In that approach, modeling languages are considered as "schemas" that can be tailored and transformed. Those schemas are artifacts in an agile software development process. Other works deal with specific contexts, such as Ringert et al. (Ringert et al. 2017), which focuses on agile MDD methods for cyber-physical systems. The students developed complex robotics applications through SCRUM and conceptual modeling. In both works (Ghiran et al. 2020) and ((Ringert et al. 2017), the final code system is generated and tested to check the teaching results.

Other works summarize teaching experiences with MDD and conceptual modeling from a wide variety of contexts. Daun et al. (Daun et al. 2017) describe the experience of teaching conceptual modeling in an online course through lecture-style videos and whiteboard-style videos. The main important challenges were the lack of interaction among students and teachers and the need to discuss and provide feedback to students. The work of Cabot and Kolovos (Cabot and Kolovos 2016) and the work of Hamou-Lhadj et al. (Hamou-Lhadj et al. 2009) present their experience of teaching MDD to undergraduate students (most papers focus on Master's students). Both works explain the basis of model-to-model and model-to-code transformations through OCL and ATL. The work of Lim (Lim 2019) describes a course to teach MDD to develop embedded software. The course is based on Class Diagrams and State Charts, while the code is manually written from those models. The work of Kuzniarz and Martins

(Kuzniarz and Martins 2016) describes the experience of a course to teach MDD with the aim of generating code from UML models. The course is taught from a theoretical perspective; the students do not apply actually transformation rules. Muller (Muller 2015) summarizes the challenges of teaching conceptual modeling to practitioners in companies. The most critical challenges are: the multi-disciplinary profile of the models; the need for exploring the customer context; modeling the dynamic behaviour of the system and moving from the technical design to the context with humans. The work of Porüban et al. (Porubän et al. 2015) describes the teaching guidelines used in teaching MDD to solve specific problems in the development process. The course is based on Domain-Specific Languages (DSLs) to generate code only for CRUD operations. The work of Eckert et al. (Eckert et al. 2016)) deals with the advantages and disadvantages of MDD in the context of a teaching course. The course is based on UML diagrams that generate code in JAVA through the Astah framework.

As conclusions of our analysis of these related works, we can highlight some assumptions: (1) All of the analyzed teaching techniques evaluate models instead of the quality of the generated code; (2) At best, some of the existing works generate small chunks of code or only CRUD functionalities (e. g., (Porubän et al. 2015), but they do not produce a fully functional system; (3) There are works e. g. (Cabot and Kolovos 2016) and (Hamou-Lhadj et al. 2009) where students must define the transformation rules and apply them, which reduces effectiveness in the system development.

In summary, we can state that there are no works that compare both a traditional method and an MDD method. Most of the existing MDD courses focus on evaluating models, which hinders the potential of the MDD paradigm in code generation. The few works that evaluate generated code do not deal with fully functional systems but with small chunks of code. The following section presents our approach to closing these gaps.

3 Teaching Methodology

The course where the experiment is conducted focuses on the comparison of MDD and a traditional development method with the aim of determining the pros and cons of MDD for the students. This way, students can judge first-hand if MDD can be applied in real software developments. At first glance, students tend to refuse the assumption that MDD is better than a traditional method or even comparable to each other. Most students think that MDD can only generate small chunks of code, as CASE tools do. During their previous undergraduate studies, they focused their abilities on programming; conceptual models are relegated to the use of UML models for documentation. Fig. 1 describes the teaching methodology that we used in our practical experience to compare MDD and a traditional method. The teaching methodology is based on problem-based learning (Perrenet et al. 2000); students have to develop problems to learn the theoretical concepts described above.

The **first** step of our methodology consists of obtaining the profile of each student. At first glance, we can claim that all of the subjects have a large background in traditional development. These students have been learning programming languages for four years in the undergraduate program as a minimum. Most of them are already working in companies in the role of developers, so we think that these students can be considered good programmers. We need to check this assumption using a demographic questionnaire. We ask about their knowledge of programming and the knowledge of MDD and the roles that they have played in the industry before attending the Master's program, and their average marks in the undergraduate program. This questionnaire is completed during the first 5 minutes of the first class.

The **second** step consists of an introduction to the MDD paradigm independently of any tool. The teacher describes the concept of MDD considering the differences between the problem space and the solution space. The teacher highlights that, in MDD, all the analysts' efforts are focused on

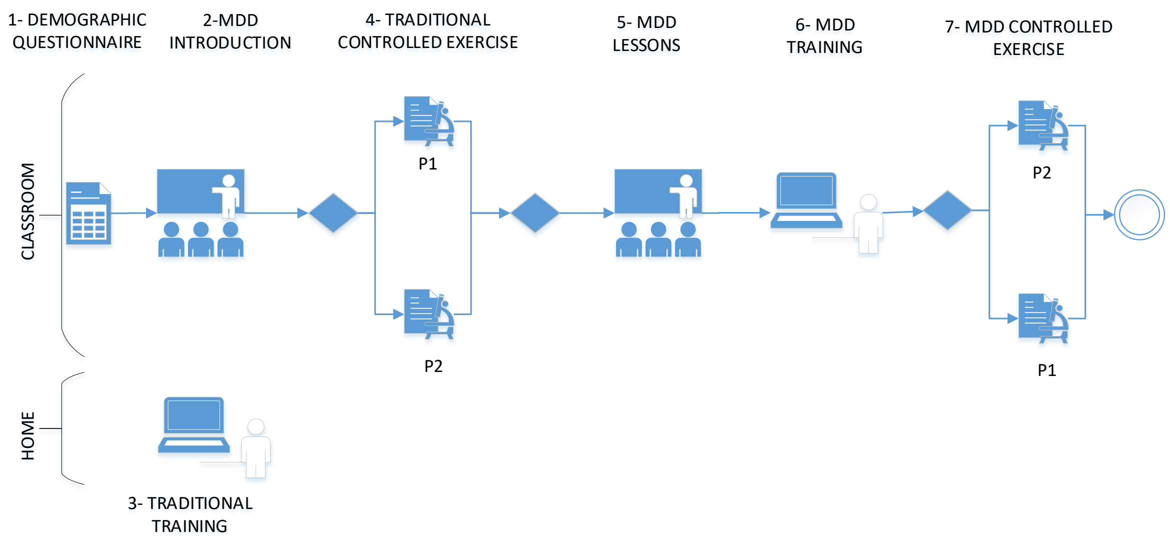

Figure 1: Steps of the teaching methodology

the problem space, whereas in traditional software development, efforts are focused on the solution space. This introduction requires two lessons (4 hours).

The **third** step is done in parallel with Step 2. While the teacher introduces MDD, the students also have one week to train in the development of a Web page application using a traditional method. Even though we can assume that all of the students are good programmers in the traditional method, we must ensure that all of them have enough knowledge to develop a system from scratch. This is why we ask students to implement a software system with a traditional method as training before doing a controlled exercise in the classroom. Since the traditional development method is beyond the scope of the course, the students perform their training as homework. Students can develop the system in the programming language that they prefer; we recommend Web development since the MDD tool generates code for the web. This way, the students can compare a traditional method with MDD in the context of Web application development. The training problem is a system for managing the routes of public buses.

The **fourth** step consists of the development of a system from scratch using a traditional method. This is a controlled exercise in the classroom. By controlled we mean that it cannot be done as homework; it must be done under teacher supervision. Once the training of Step 3 has been completed, we can ensure that the students have enough knowledge to develop a system with their favorite programming language. We divide the students into two groups; half of them develop Problem 1 and the other half develop Problem 2. **Problem 1** (Invoice Problem) aims to manage an electrical appliance company. Once the repair is done, the system must create an invoice. **Problem 2** (Photography Problem) aims to manage a company that works with freelance photographers. The system must register who is the owner of each photo and the amount of money to pay to each photographer. Both problems have similar complexity; Problem 1 has 272 function points and Problem 2 has 199 function points. The reason for using two different problems is to generalize the results of the experience since we make the results independent of any problem. This exercise requires two classes (4 hours). Both problems are too complex to be completed in such a short period of time, but we just want to see what percentage of the system is developed. None of the students finished either problem completely.

The **fifth** step is teaching MDD applied to a specific tool. In our course, we teach WebRatio (Brambilla and Fraternali 2014), which is an MDD tool that works with different models: (1) a *Domain Model*, which is a conceptual schema that represents all of the classes and their properties and relationships needed in the system represented in UML Class Diagram notation; (2)the *Interaction Flow modeling Language (IFML)* (OMG 2021), which represents the system interfaces (adopted by the OMG as the standard to represent interfaces abstractly); (3) the *Action Model* that specifies the CRUD (Create, Read, Update, Delete) operations that can be done in the system. We need five classes (10 hours) to teach WebRatio to students.

The **sixth** step is training the students in the MDD method, similar to the training done in Step 3. In this case, we aim to train the students in the MDD method. The main difference regarding the training of the traditional method is that, in this step, the training is done in the classroom with the support of the teacher. Note that this is the first time that most of the students have worked with MDD, so the teacher must be close-by in order to resolve questions that arise. For the training, we used a system to manage films renting video club. Three classes (6 hours) are required to complete the training. Students that need more time can finish the development as homework.

The **seventh** step is a controlled exercise in the classroom to develop a system from scratch using MDD. The problems used are the same problems as those used in Step 4, but the problems are swapped. The students who developed Problem 1 in Step 4, now develop Problem 2 in this step (and vice versa). Swapping problems reduces the learning effect (carrier effect) between development methods. The students do not know the problem since this is the first time that they must develop it. Two lessons (4 hours) are required to develop the assigned problem. As in Step 4, the problems are too large to be completed in such a short time. We aim to check which percentage of the system's students can complete in the four hours.

Note that all of the steps except for Step 3 are conducted in the classroom with the support of the teacher. The course was designed to learn and work mainly in the classroom. To ensure that students do not work on the controlled problems as homework, we collect the problem requirements at the end of each class. Moreover, the virtual machines where WebRatio is installed are hidden when each class is over, so the students cannot access WebRatio outside of the controlled sessions.

The final mark is calculated by measuring the percentage of test cases successfully satisfied in the four developments from scratch: traditional development training, experimental traditional development, MDD training, and experimental MDD development. All of them have the same weight in the final mark. This represents 95% of the final mark. The other 5% depends on the results of the knowledge questionnaire. There is no written exam.

3.1 Bloom's Taxonomy of Educational Objectives

Bloom's taxonomy is used to classify the educational objectives that students must meet at the end of the course. Below, we describe the goals of our course:

- $G1$: Students will be able to label the different conceptual primitives of a conceptual model.
- $G2$: Students will be able to distinguish the aim of each conceptual model.
- $G3$: Students will be able to build a conceptual model in WebRatio.
- $G4$: Students will be able to analyze information represented in a WebRatio conceptual model.
- $G5$: Students will be able to evaluate the quality of a WebRatio conceptual model.
- $G6$: Students will be able to develop a fully functional system from scratch using WebRatio conceptual models.

Based on the work of Bork (Bork 2019), we classify our teaching goals into a two-dimensional schema for the assessment of educational objectives: **the knowledge dimension** and **the cognitive process dimension**.

The knowledge dimension classifies the knowledge into four categories: (1) Factual Knowledge, the basic elements that students must know; (2) Conceptual Knowledge, interrelationships between the basic elements; (3) Procedural Knowledge, how to do something; and (4) Metacognitive Knowledge, knowledge of cognition in general and knowledge of one's own cognition.

The cognitive process dimension specifies what is to be done or learned according to six categories: (1) Remember, retrieving relevant knowledge from long-term memory; (2) Understand, determining the meaning of concepts; (3) Apply, using a procedure; (4) Analyze, detecting how the parts relate each other and to an overall structure; (5) Evaluate, making judgments; (6) Create, putting elements together to form an original product. The results of classifying our goals into both dimensions are shown in Tab. 1.

4 Experiment Design

This section describes the design of an experiment that we conducted in the 2020/2021 course. We first describe profile of the participants, and then we describe the hypotheses, variables, and metrics.

4.1 Subjects

The subjects were 22 students of the Master's program in Engineering and Software Systems Technology at Universitat Politècnica de València (UPV). They worked in pairs (except for two students who worked alone) and had to develop the training problems and the controlled problems by working together. This decision was due to space restrictions. Moreover, we have seen in previous years that working in pairs involves more collaboration and the results tend to be better than when working individually.

Now, we analyze the results of the demographic questionnaire extracted from Step 1. Fig. 2 shows the maximum programming experience of the students. The figure shows that there are 10 students with more than one year of experience, 5 students with less than one year of experience, and only 7 students with no experience. Fig. 3 shows the roles of students in a company. It can be observed that most students (11) were junior programmers. This means that they work daily implementing code manually. Finally, Fig. 4 shows the knowledge of MDD that the students had before attending the course. The figure shows that 16 students had no idea of MDD or had just heard some ideas. Only 3 students took lessons before this course, and only 1 has worked occasionally with MDD. These results confirm our assumption that students are good programmers, but they have no previous knowledge of MDD. At first glance, this leads us to think that students will be more motivated to program than to use modeling, which is the goal of our course.

Tab. 2 shows the average marks (each mark is between 0 and 10) of the students in the undergraduate program before starting the Master's course as well as the final marks obtained in the MDD course. This is helpful for assessing the level and aptitude of students. It can be used to look for a correlation between the marks on previous courses and marks on the modeling course (analyzed in the results section).

4.2 Hypotheses, Variables and Metrics

This section describes the hypotheses we aim to check with the students, the variables we use to check such hypotheses, and the metrics used in the variables. We are proposing this experience because we aim to improve the motivation for learning MDD, to improve the knowledge of MDD, to ensure that students have learned educational competences to develop quality software through MDD, and to improve their satisfaction of working with MDD. These objectives lead to the definition of the following null **hypotheses**:

- H_{01}: The attitude towards using MDD in software development is the same before and after the course.

Table 1: Bloom's Taxonomy of teaching goals

	Remember	Understand	Apply	Analyze	Evaluate	Create
Factual Knowledge	G1					
Conceptual Knowledge		G2				
Procedural Knowledge			G3			G6
Metacognitive knowledge				G4	G5	

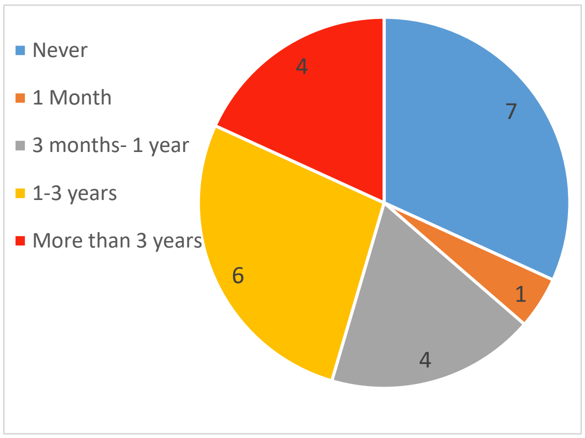

Figure 2: Years of experience of programming in a real company

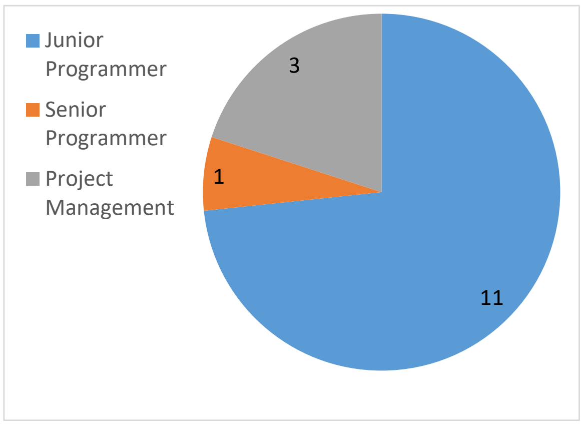

Figure 3: Roles played in a real company

- H_{02}: The knowledge of MDD is the same before and after the course.
- H_{03}: The quality of the software developed using MDD is the same as the software developed using traditional development.
- H_{04}: The satisfaction of developing software using MDD is the same as the satisfaction of using traditional development.

To analyze these hypotheses, we need **variables** (factors and response variables) (Juristo and Moreno 2001) and their **metrics**. Factors are variables that affect the response variables that we want to understand. We have two factors in our

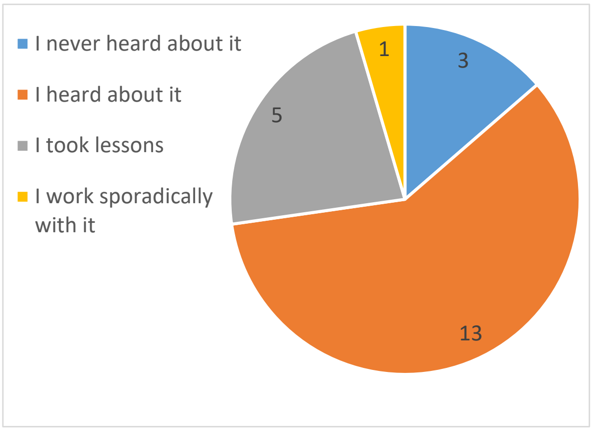

Figure 4: Knowledge of MDD before the lessons

Table 2: Students' marks before and after the MDD course

Student	Average of marks before the course	Marks of the course
1	6	8
2	7.5	9.5
3	7	8.5
4	9.1	7.5
5	8.5	9.2
6	6.4	9
7	8.02	9.5
8	7.8	6
9	7.6	7.5
10	7.6	8.5
11	7.4	8.6
12	7.9	9.5
13	7.7	9.2
14	6.2	8.6
15	7.4	6
16	7.2	9.5
17	9	8.5
18	7	8.5
19	7.3	9
20	7	9
21	8.7	9
22	7.6	9

validation. One factor is the use of the teaching methodology, with two levels: before and after teaching. The other factor is the development method, with two levels: MDD and a traditional method.

Response variables are the effects studied in the experiment caused by the manipulation of factors. We have a response variable for each hypothesis. H_{01} is measured through the variable *Attitude*. By attitude, we mean the opinion or feeling that students have about MDD (Sitaraman et al. 2002). The metric that we use for this response variable is obtained from a questionnaire defined by us with a 5-point Likert scale (from 1 to 5 points). Tab. 3 shows the questions that compose the questionnaire. Questions AQ1, AQ2, and AQ3 deal with how confident the student feels with MDD. Questions AQ4 and AQ5 obtain the student's opinion of the code generation process. Question AQ6 asks about the student's satisfaction with using MDD. Questions AQ7, AQ8, and AQ9 deal with how easy MDD is. Finally, Question AQ10 deals with the usability of the tools that support the MDD method. The Attitude metric is the sum of the answers to all of these questions. Thus, we have a single value for Attitude for each student. Possible values for Attitude fluctuate between 10 and 50 (each question individually fluctuates between 1 and 5).

Table 3: Questionnaire for Attitude towards MDD

ID	QUESTION
AQ1	MDD is a challenging activity.
AQ2	If I work for a company that uses MDD, I feel qualified to develop a real software system.
AQ3	I feel capable of working in MDD with models that are different from the models seen in this course.
AQ4	MDD allows code to be generated with fewer errors than a traditional method.
AQ5	MDD allows code to be generated in less time than a traditional method.
AQ6	I am more satisfied working with MDD than with a traditional method.
AQ7	MDD notation allows me to identify errors easier than a traditional method.
AQ8	The effort required to learn MDD is less than the effort needed to learn a traditional method.
AQ9	The effort required to use MDD is less than the effort needed to use a traditional method.
AQ10	The usability of an MDD tool is better than the usability of a tool of a traditional method.

For H_{02}, we have defined the response variable *Knowledge*, which means how much the student knows about MDD. The metric for this response variable has also been obtained from a questionnaire with four options where only one is correct. Tab. 4 shows the topic of the six questions that compose the questionnaire. We have two questions related to the Class Model (KQ1 and KQ2), two questions related to the Interaction Model (KQ3 and KQ4), and two questions related to the Actions Model (KQ5 and KQ6). The metric for Knowledge is the sum of all of the correct answers of this questionnaire. Thus, we have a single value for Knowledge for each student. Possible values for Knowledge fluctuate between 0 and 10 (each question is weighted the same, 1.66). Note that from a teaching point of view, in order to ensure that students have enough knowledge of MDD, we not only use these questionnaires, but we also use the development of the system from scratch.

Table 4: Questionnaire for Knowledge

ID	QUESTION
KQ1	Inheritance relationship between classes.
KQ2	Derived attributes of a class.
KQ3	Menus in interfaces.
KQ4	Navigation among interfaces.
KQ5	modeling actions.
KQ6	Relationship among forms and actions.

For H_{03}, we have defined the response variable *Quality*. IEEE defines quality as "the degree to which a system, component, or process meets specified requirements" (IEEE 1991). To measure this response variable, we have prepared a suite of test cases to be checked in both problems. Each functional requirement (described in Panach and Pastor 2021b) has been transformed into a test case that must be run on the executable system. Each test is composed of several sequential items that are required to accomplish the target of the test. Tab. 5 shows an example of the test case for creating a repair card. In general, in each test case, there are items for checking the normal execution, such as the last item in Tab. 5. The test uses the percentage of items successfully passed as possible values. Therefore, possible values for each test is a range between 0% (no test passed) and 100% (all tests passed). These tests are run on the final software systems developed. The Quality of the system is calculated as the average of all of the run tests. In summary, the quality of the developed systems depends on the percentage of functional requirements that the subject managed to satisfy during the experimental session. The students work in pairs in the classroom, which means that we have one value of Quality for each pair of students.

For H_{04}, we have defined a satisfaction questionnaire with a 5-point Likert scale (from 1 to 5 points). The metric used to measure this variable is a satisfaction questionnaire that was built using the framework developed by Moody (Moody 2003). Moody defined a framework to evaluate model quality in terms of Perceived Usefulness

Table 5: Example of test case

Create a repair card
Insert identifier: 1
Insert customer passport no.: 33472035L
Insert repair date: 21/02/2021
Insert description: TV repair
Insert amount due: 100€
Insert technician's name: Francisco García
Check that all of the above data has been saved in the system

Table 6: Questionnaire for Satisfaction

ID	QUESTION
SQ1	I found the procedure for applying MDD to be simple and easy to follow.
SQ2	I think MDD reduces the effort required to develop web applications.
SQ3	The conceptual model of the Web applications developed with MDD is easily understood and modifiable by other developers.
SQ4	Overall, I find MDD easy to use.
SQ5	MDD makes it easy for the developer to fix web application bugs.
SQ6	You could easily explain MDD to someone else who didn't know you.
SQ7	Overall, I found MDD helpful.
SQ8	In general MDD is practical for implementing the needs of the users of the Web application.
SQ9	In my opinion, MDD is easy to use in the web application that I have developed.
SQ10	I would definitely use MDD to develop web applications.
SQ11	MDD seemed clear and easy to understand.
SQ12	Overall, I believe that MDD provides an effective solution for web application development.
SQ13	By using MDD you can build large web applications efficiently.
SQ14	I am confident that I now have the necessary skills to apply MDD in practice.
SQ15	Overall, I think MDD is an improvement over other development methods.
SQ16	I will preferably try to use MDD over other development methods if I have to develop other web applications in the future.

(PU), Perceived Ease Of Use (PEOU), and Intention To Use (ITU). This framework has been previously validated and is widely used. According to Moody (Moody 2003), we defined eight questions to measure Perceived Usefulness, six questions to measure Perceived Ease of Use, and two questions to measure Intention to Use. Tab. 6 shows the questions that compose the satisfaction questionnaire. These questions are for measuring the satisfaction with MDD; we have a similar questionnaire for measuring the satisfaction with the traditional method. Questions SQ1,SQ4, SQ6, SQ9, SQ11, and SQ14 are for measuring Perceived Ease of Use; questions SQ2, SQ3, SQ5, SQ7, SQ8, SQ12, SQ13, and SQ15 are for measuring Perceived Usefulness; questions SQ10, and SQ16 are for measuring Intention to Use. The metrics for Perceived Ease of Use, Perceived Usefulness and Intention to Use are the sum of their respective questions. After calculating this sum, we have a single value for Perceived Ease of Use, Perceived Usefulness, and Intention to Use for each student. Possible values fluctuate between 6 and 30 for Perceived Ease of Use, 8 and 40 for Perceived Usefulness, 2 and 10 for Intention to Use. The satisfaction questionnaires are completed after applying the traditional method and MDD respectively, before discussing the results for quality assessment in the last session.

In addition to the questionnaires regarding the hypotheses, at the end of the course, we asked the students about the pros and cons of using MDD. We asked the questions directly in an open-question format. The analysis of these questions allows us to explain the numerical data results of the hypotheses.

The validation follows a within-subjects design since we apply both treatments to all of the students. We obtained data from all of the subjects before and after the course (the Use of the Teaching Methodology factor), and after applying each development method (the Development Method factor). The Use of the Teaching Method factor answers H_{01} and H_{02}, while the Development Method factor answers H_{03} and H_{04}. For the Development Method factor, we have a block-variable named Problem to represent the problems in which treatments are applied. We blocked by the problem to avoid the learning effect of using the same problem in both methods (traditional and MDD). This means that we analyze the interaction of Development Method*Problem, but we are not interested in analyzing the problem independently of the development method.

4.3 Experiment material

This section describes the material used in the experiment. Even though the problems may have several possible solutions, we describe the experimenters' solutions. Fig. 5 shows the UML Class Diagram for the Invoice Problem. The system must create and save repair orders for technical issues. These repairs are done by technicians that go to the customer's house to make the repair. Once the repair is completed, the technician must create an invoice to collect money from the customer. Some repairs are randomly audited. We need to register who did the audit, the company where the auditor works, and the material used in the audit. Apart from repairing technical issues, the company also offers courses for the customers. These courses aim to teach customers ways to resolve technical issues and make repairs on their own in the future. The task is to manage the details of the courses, including the subjects that compose the courses and the teachers involved.

Fig 6 shows the UML Class Diagram for the Photography Problem. The system manages the information of candidates who want to be photographers and photographers already been hired. The applicants candidates can submit their curriculum, and a group of representatives of the company must decide whether or not to accept them. Each accepted photographer has a classification that is used to determine a price specific per photo. This category can increase as the experience of the photographer increases. There are two types of jobs that a photographer can do: stories and scoops. Stories require a publisher to pay for the photos immediately, whereas a scoop requires a delivery order because they are paid later. The price that the photographer receives for each photo on the report depends on her/his classification, whereas the photographer can define the price to receive for a scoop.

Fig. 7 shows a small part of the Interaction Model in WebRatio to represent the interface of the Invoice Problem to list and create invoices. Fig. 8 shows the Action Model of the Invoice Problem that represents how to create a course. The description of the problems as they were shown to subjects is presented in (Panach and Pastor 2021b). The repository also includes a possible solution for both problems to be opened using WebRatio.

4.4 Threats to Validity

This section describes the threats addressed by the experiment setting. We have classified the threats according to the classification provided by Wohlin et al. (Wohlin et al. 2012). We classify the threats into four groups: Conclusion Validity, Internal Validity, Construct Validity, and External Validity.

Conclusion Validity: This type of threat deals with the ability to draw the correct conclusion about the relations between the treatment and outcome. Our experiment may suffer *Low statistical power*, which means that the sample size is not large enough to reject the null hypotheses. According to G*Power (Erdfelder et al. 1996), for a moderate effect size (0.5) in a within-subjects design, we need a sample size of over 16 subjects to get good values of power (0.96). We have 22 subjects, so we can state that even though our sample is not large, we mitigated this threat. Another mitigated threat is *Subjects of random heterogeneity*, which means that the subjects are randomly selected and their background is too heterogeneous. We managed to mitigate this threat using training sessions in a traditional method and in MDD. This way, we could ensure that the students had enough knowledge to follow the course. Another threat that the experiment may suffer is *Reliability of measures*, which means that there is no guarantee that the outcomes will be the same if a phenomenon is measured twice. We mitigated

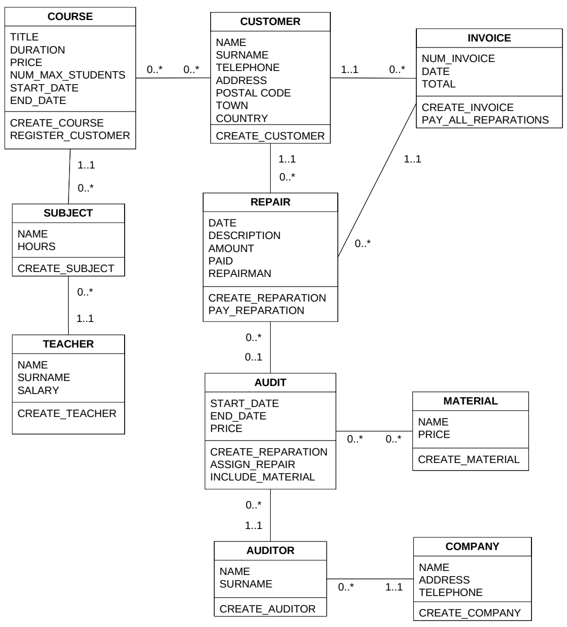

Figure 5: UML Class Diagram for the Invoice Problem

this threat by applying the metrics objectively by one experimenter.

Construct Validity: This type of threat concerns generalizing the result of the experiment to the concept or theory behind the experiment. The validation may suffer *Evaluation apprehension*, which means that the students are afraid of being evaluated. We did not manage to mitigate this threat since experimental tasks are exercises that the students had to complete in order to pass the course. Another threat that the validation may suffer is *Interaction of different treatments*, which means that there is no way of concluding whether the effect is due to either of the treatments or to a combination of several treatments. We mitigated this threat by considering two factors in separate response variables, but other factors that were not considered in the design may have affected the results.

Internal Validity: This type of threat deals with influences that can affect the factor concerning causality. A threat that may impact the validation is *Learning of objects*, which means that the students may learn things with the first problem that help them in the second problem. We mitigated this threat by using two different problems to measure the Development Method factor. Another threat is *Subjects' motivation*, which means that less motivated pairs of students may achieve worse results than highly motivated pairs. We did not manage to mitigate this threat.

External Validity: This type of threat concerns conditions that limit our ability to generalize the results of our experiments to industrial practice. Our validation may suffer *Object dependency*, which means that results may depend on the objects used in the experiment and they cannot be generalized. We mitigated this threat by using two problems for the Development Method factor. Another threat is *Interaction of history and treatment*, which means that treatments are applied on different days and the circumstances on that day affect the results. We mitigated this threat by applying the treatments in the same schedule and context.

5 Results of the Validation

This section describes the analysis of the results obtained from the metrics defined previously. The

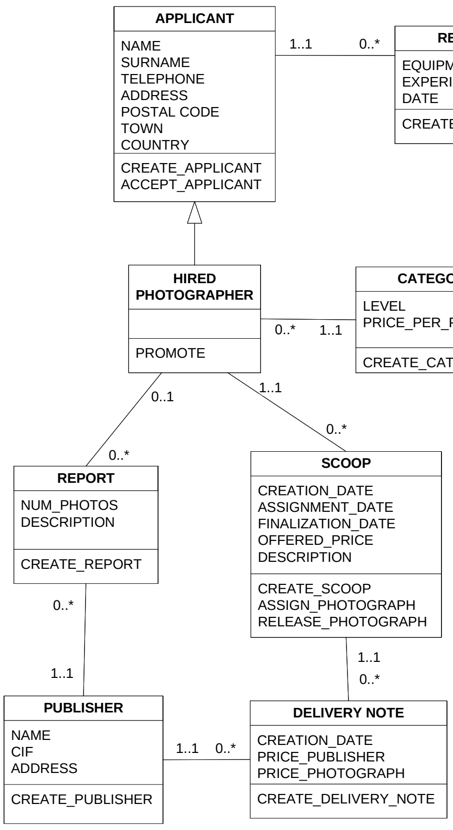

Figure 6: UML Class Diagram for the Photography Problem

raw data is presented in (Panach and Pastor 2021b). The data is analyzed through a statistical test named Mixed Model to look for significant differences. We have chosen a Mixed Model (West et al. 2014) because when we analyze the Development Method factor, we also need to include the block variable (Problem), and both elements are repeated measures. The Mixed Model is a test that allows more than one repeated measure to be analyzed, so it is the most suitable for our design. We have considered significant differences when the p-value > 0.05.

We calculated the effect size for only those response variables with significant differences using Cohen's d. The effect size is used to analyze the magnitude of such differences. Cohen's d is defined as the difference between two means divided by a standard deviation of the data. According to Cohen (Cohen 1988), the meaning of the effect size is as follows: more than 0.8 is a large effect; from 0.79 to 0.5 is a moderate effect; from 0.49 to 0.2 is a small effect.

We have also reported the descriptive data to analyze which treatment yields better results using Box and Whisker plots. Finally, we have analyzed a possible correlation between the final students' marks with the values of our response variables after using MDD with Pearson correlation. Values less than 0.3 mean no correlation, values between 0.3 and 0.5 mean a weak correlation, values between 0.5 and 0.7 mean a moderate correlation, and values larger than 0.7 mean a high correlation. P-values lower than 0.05 mean that the correlation is significant, i. e., we have a sample size that is large enough to guarantee the result of the cor-

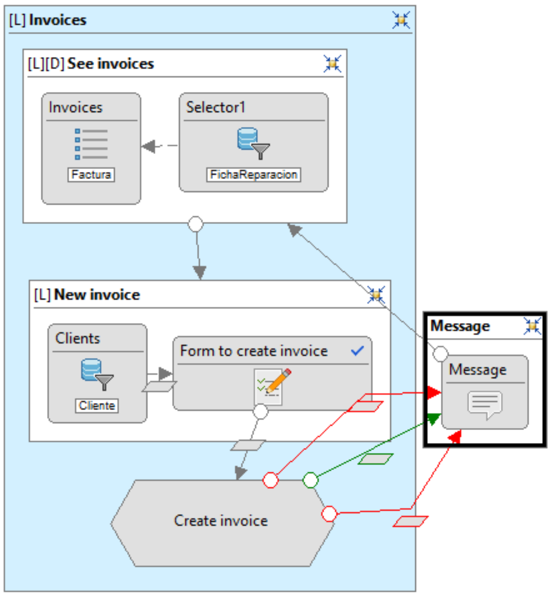

Figure 7: Excerpt from the Interaction Model of WebRatio in the Invoice Problem

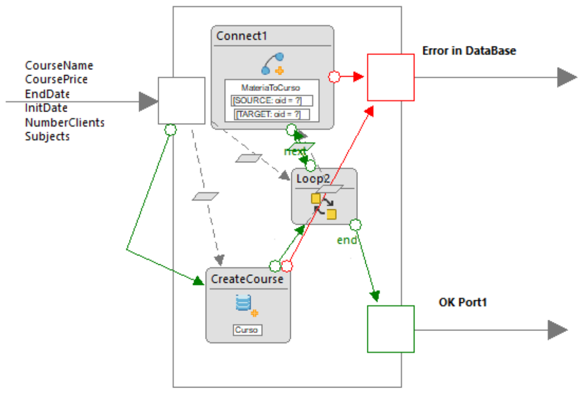

Figure 8: Excerpt from the Action Model of WebRatio in the Invoice Problem

relation. Below, we present the results for each response variable.

Tab. 7 shows the p-value of the Mixed Model and the effect size for the **Attitude** response variable. This response variable is calculated for the Use of the Teaching Methodology factor taking into account each student individually (we have the metric for each student). Since p-value=0.00, we can state that there are significant differences in the treatment Use of the Teaching Methodology. The effect size shows that these differences are moderate. Figure 9 shows the Box and Whiskers plot for Attitude. It can be observed that the first quartile, median and third quartile are better after using our teaching methodology. Therefore, we can reject the null hypothesis H_{01} and state that our course involves a change in the student's attitude toward the use of MDD. Students are more open to working with MDD after attending the course.

We also aim to look for a possible correlation between the attitude after attending the MDD course and the marks obtained at the end of the course. Pearson correlation yields -0.218, which means no correlation between the two variables. Significance yields a p-value of 0.331, which means the sample size does not have enough statistical power to look for correlations. Therefore, we can conclude that students' attitude does not affect their marks.

Table 7: Statistics for Attitude

Factor	p-value	Effect Size
Use of Teaching Met.	0.00	0.56

Tab. 8 shows the statistical results for the **Knowledge** response variable using the Mixed Model. This response variable is calculated for the Use of the Teaching Methodology factor taking into account each student individually (we have the metric for each student). The P-value is less than 0.05, which means that there are significant differences between using our teaching methodology and not using it. The effect size shows moderate differences. Fig. 10 shows the Box and Whiskers plot for Knowledge. It can be observed that the first quartile, median, and third quartile have better results after the course. Therefore, we can reject the null hypothesis H_{02} and state that the use of our teaching methodology involves changes in the knowledge of MDD. The plot shows that after the course, most of the students achieved a mark in knowledge close to 10 points (the maximum possible), which means that the students learned all of the MDD concepts.

Table 8: Statistics for Knowledge

Factor	p-value	Effect Size
Use of Teaching Met.	0.00	0.70

Using Pearson correlation, we aim to look for a possible correlation between the knowledge after attending the MDD course and the final marks. The results show a correlation of 0.041, which means there is no correlation between the two variables. Significance has a p-value of 0.856, so the statistical power is not enough to look for correlations. Note that, even though the final mark depends on the questionnaire for knowledge, this questionnaire represents only 5% of the final mark.

Since the demographic questionnaire collects the average of the student's marks in the undergraduate program studied before attending the Master's course (Tab. 2), we analyze a possible correlation between these previous marks and the mark in this MDD course. The results of the Pearson correlation yield -0.024, which means that there is no correlation. Moreover, the p-value of significance is 0.916, so the sample size is not large enough to conclude a possible correlation between the two marks. We can state that the average of marks obtained in previous courses are not correlated with marks obtained when learning MDD. Therefore, previous knowledge is not related to success in using MDD.

Tab. 9 shows the statistics for **Quality** using the Mixed Model. Note that this response variable is calculated for the Development Method factor and the Problem block variable. The sample unit for this variable is the pair of students since we have data of this metric for each pair. We have analyzed the Development Method and the interaction Development Method*Problem. Only the Development Method yields significant results, which means that there are differences between the two development methods independently of the problems. The effect size shows a low effect. Fig. 11 shows the Box and Whiskers plot for Quality. The results show better quality in MDD for the first quartile, median, and third quartile. This means that the students working with MDD pass more tests than the same students working with a traditional method. Therefore, we can reject the null hypothesis H_{03} and state that there are significant differences in Quality depending on the development method used.

If we analyze the correlation between quality in MDD and final marks, the Pearson correlation yields 0.723. This is a strong correlation, which

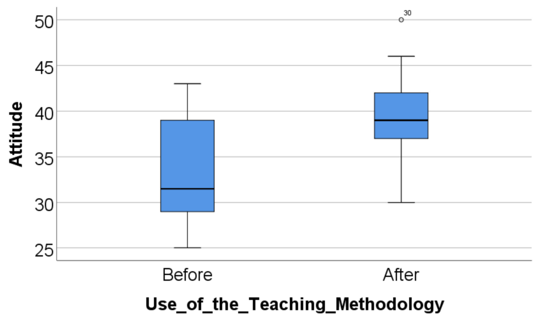

Figure 9: Box and Whiskers plot for Attitude

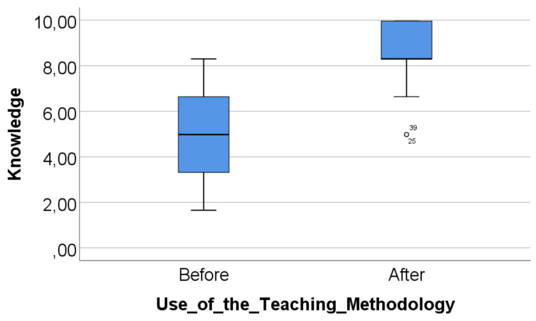

Figure 10: Box and Whiskers plot for Knowledge

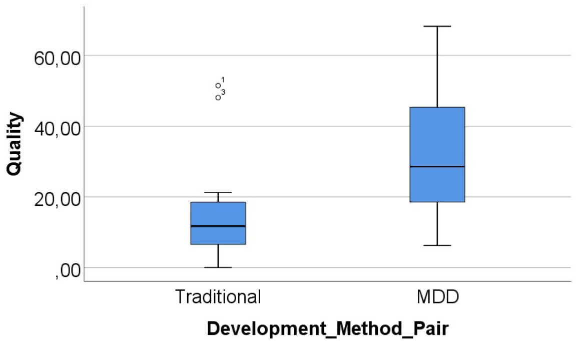

Figure 11: Box and Whiskers plot for Quality

means that students with the best quality in their MDD models obtain the best marks. Note that even though the quality of the MDD development is used to calculate the final mark, there are other developments used in the calculus (the traditional development and both trainings). The significance of the correlation yields a value of 0, which means that we have a sample size that is large enough to

Table 9: Statistics for Quality

Factor	p-value	Effect Size
D. Method	0.04	0.39
D. Method*Problem	0.43	-

certify the correlation between the two variables. This result indicates that the students with the best marks are those that developed the best systems with MDD.

Tab. 10 shows the p-value using the Mixed Model and the effect size for **Satisfaction** measured in terms of Perceived Ease of Use, Perceived Usefulness, and Intention to Use. These results have been calculated considering the Development Method as a factor and the Problem as a blocking variable. The sample units are the subjects since we have metrics for each subject. In this case, we have significant results for Perceived Ease of Use and Intention to Use, whereas Perceived Usefulness shows no significant differences. Both significant results show a low effect size. Moreover, we have a significant result for the Intention to Use*Problem interaction, which means that there is a problem that yields better results for one of the treatments.

The Box and Whiskers plots of Perceived Ease of Use in 12 show that the first quartile, median, and third quartile are better for MDD. The medians of Perceived Usefulness in Fig. 13 show no differences between the two treatments since they are at the same level, even though the first quartile and third quartile are slightly better for MDD. Fig. 14 shows that the traditional method yields better results than MDD for Intention to Use. The first quartile, median and third quartile are better for the traditional method. Fig. 15 shows the profile plot to interpret the significant differences of the Development Method*Problem interaction for Intention to Use. We see that Problem 1 has a better Intention to Use for the traditional method rather than for MDD. This difference does not appear in Problem 2. This could be because Problem 2 included an inheritance that Problem 1 does not have. This could slightly increase the difficulty of Problem 2, masking possible differences between the Intention to Use of the two development methods in this problem. According to all of these results, we can reject H_{04} for both Perceived Ease of Use and Intention to Use. Perceived Ease of Use shows better results for MDD, whereas Intention to Use shows better results for the traditional method.

Table 10: Statistics for Satisfaction

Metric	Factor	p-value	Effect Size
Perc. Ease of Use	D. Method	0.03	0.29
	D. Method*Problem	0.14	-
Perceived Usefulness	D. Method	0.11	-
	D.Method*Problem	0.11	-
Intention to Use	D. Method	0.04	0.27
	D. Method*Problem	0.03	-

An analysis of the Pearson correlation between the final marks and the three metrics for satisfaction provides the following values: Perceived Ease of Use yields a correlation of 0.315 (with a p-value of 0.153); Perceived Usefulness yields -0.113 (with a p-value of 0.616); and Intention to Use yields -0.415 (with a p-value of 0.055). The only satisfaction metric with a correlation is Intention to Use, with a weak correlation. The p-value indicates that the sample size is not large enough to generalize this conclusion. This result means that, even though it is not significant, the students with the best marks do not have the intention to use MDD in the future.

6 Discussion

This section explains the outcomes that the validation yields, combining numerical results with the pros and cons that the students discussed as open questions at the end of the course. Below, we discuss the results for each response variable.

With regard to **Attitude**, we can conclude that after the course, the student's attitude towards working with MDD was significantly higher. At the beginning of the course, the students were reluctant to use MDD, but this perception completely changed when they compared their outcomes with

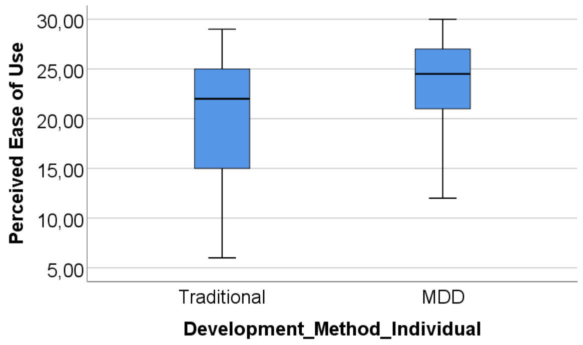

Figure 12: Box and Whiskers plot for Perceived Ease of Use

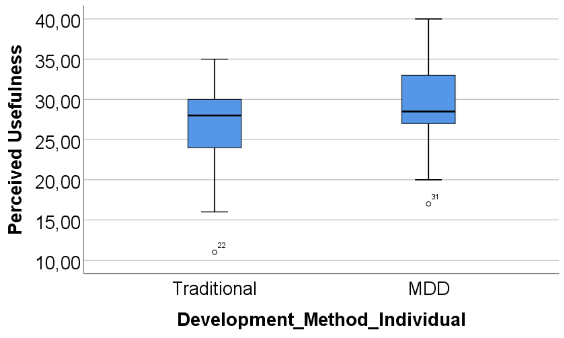

Figure 13: Box and Whiskers plot for Perceived Usefulness

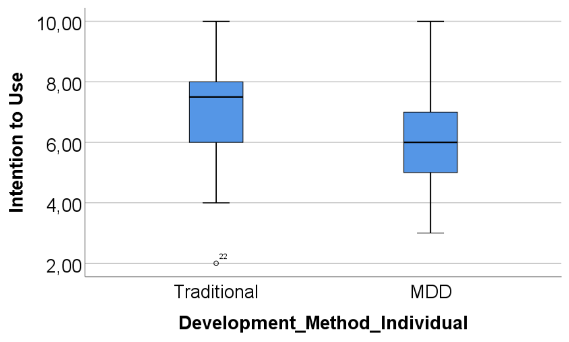

Figure 14: Box and Whiskers plot for Intention to Use

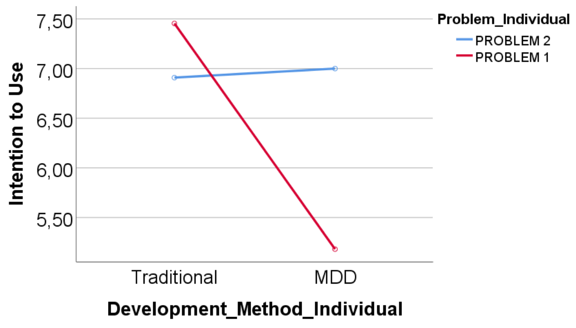

*Figure 15: Profile plot for Development Method*Problem*

the two development methods. Some of the pros mentioned by the students about their attitude towards MDD are: the time required to develop a system with MDD is less than with a traditional method; the deployment time of the system with MDD is easier than with a traditional method; the migration to other platforms is easy thanks to model-to-code transformations. There are also some cons regarding attitude after attending the course: it is not easy to identify how model characteristics affect the generated code; MDD does not allow as much flexibility as in the development with a traditional method.

With regard to **Knowledge**, the results show that after the course, the students had more knowledge of MDD than before the course. Even though there were models that were well known by the students before the course, such as the UML Class Diagram, other models such as Interaction Model (IFML) and Action Model had to be learned from scratch during the course. The questionnaire for measuring Knowledge before the course included two questions about the UML Class Diagram (KQ1 and KQ2 in 8). KQ1 was about inheritance in a class, where all of the students answered the questions correctly. KQ2 was about derived attributes in a class, where 17 (of 22) students answered correctly. However, most of the students did not manage to answer questions regarding IFML and the Action Model correctly before the course. Only after finishing the course did all of the students manage to answer all of the knowledge questions successfully. One of the pros of MDD for Knowledge is that it is easy to learn the conceptual models that support the MDD development. One of the cons of MDD for Knowledge is that the tool that supports the MDD method is not very usable, which hinders more comfortable learning.

With regard to **Quality**, we can state that the quality of systems developed using MDD is higher than the quality of systems developed using a traditional method. This means that during the four hours spent on the development, the students who used MDD managed to fulfil more requirements than in the same period of time with a traditional method. The students stated the following pros of MDD regarding quality: Small changes in the models allow a quick correction of the system; The generated code is free of errors; It is easy to change the interface to fulfil the requirements. As cons for quality, the students described the following: There are limited layout templates for the graphical user interface; Errors in the models are propagated to errors in the code. We also found a correlation between final marks and the quality of the MDD developments. This means that the teaching methodology manages to teach MDD to students in such a way that they can develop a fully functional system from scratch.

With regard to **Satisfaction**, only Perceived Ease of Use was significantly better in MDD than in the traditional method. This means that even though the background in conceptual models was limited to UML Class Diagrams, the students perceived MDD as being very easy to use. They did not consider the learnability of the MDD paradigm to be a problem. Perceived Usefulness showed no difference between MDD and the traditional method. This means that the students put both development methods on the same level. The fact that the students considered both development methods to be at the same level means that, although they are good programmers, they feel qualified to work with MDD after just a single course. Intention to Use indicated that a traditional method has significantly more intention to be used. This means that, after the course, the subjects still had the intention to continue developing systems using a traditional method. This result also arises when we analyze the correlation between final marks and Intention to Use. The students with better marks had less intention to use MDD in the future. The students consider that MDD is easy and useful and develops quality systems, but they still have the intention to work using traditional development methods. This assumption can be explained by some of the cons related to Satisfaction of MDD that were discussed by the students: There is not a huge community working with MDD; Apart from CRUD operations, it is not easy to model other functionalities; Generated code is very difficult to understand, so manual code tweaking is not easy. As pros for Satisfaction of MDD, the students discussed the following: It is easier to start to develop if you have no background in a traditional method; visual models help in the understandability of the system to be developed.

As academic results, we highlight that all of the students completed the four exercises that were required for the marks of the course (independently of the level of quality), so all of them passed the course.

7 Conclusions

This paper describes a practical experience of teaching conceptual modeling in an MDD paradigm. The course is part of a Master's degree program at Universitat Politècnica de València (UPV), where students are good programmers but are not motivated to learn MDD. Our teaching methodology consists of a comparison of a traditional method versus an MDD method that the students experience through problem-based learning. Students must compare a system that is developed using a traditional method for four hours versus a system that is developed with MDD requiring the same amount of time.

The validation of the teaching experience has been done based on four response variables: Attitude, Knowledge, Quality, and Satisfaction. Attitude and Knowledge show significantly that, after applying the teaching methodology, students have a more positive attitude and more knowledge of MDD significantly. The results also show that MDD produces software of significantly higher quality. Significant results for Satisfaction show that students perceive MDD as being easy to use, but they have no intention of using it in the future.

These response variables were complemented with open questions in a discussion session where students had to discuss the pros and cons of using MDD.

After the experience, we have learned the following lessons: (1) Students only notice the advantages of MDD when they compare their productivity using both development methods. When they start to learn conceptual models, they do not think that they will manage to develop systems easily. This assumption only changes when they generate code; (2) Background in a traditional method affects how students deal with some of the MDD challenges. For example, students try to apply well-known techniques such as debugging while running the system, or they frequently aim to test the code by running the system after a few changes (trial and error). Students with no idea of a traditional method may have dealt with these problems differently, paying more attention to models and not the finally generated system; (3) Students are really concerned with how graphical user interfaces show the system. In general, students are more interested in personalizing the interfaces than in ensuring the right functionality of the system. This could be one of the reasons why students do not have the intention of using MDD in the future since personalization of interfaces is not easy with MDD.

As future work, we plan to repeat the teaching methodology in the coming courses. We plan to change the tool that supports the MDD method to see if the results are independent of the tool. We also plan to repeat the experience by comparing two different MDD tools instead of comparing a traditional method versus MDD.

References

Atkinson C., Kühne T. (2003) Model-driven development: a metamodeling foundation. In: IEEE Software 20(5), pp. 36–41

Berre A. J., Huang S., Murad H., Alibakhsh H. (2018) Teaching modelling for requirements engineering and model-driven software development courses. In: Computer Science Education 28(1), pp. 42–64

Bork D. (2019) A Framework for Teaching Conceptual Modeling and Metamodeling Based on Bloom's Revised Taxonomy of Educational Objectives. In: Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)

Brambilla M., Fraternali P. (2014) Large-scale Model-Driven Engineering of web user interaction: The WebML and WebRatio experience. In: Science of Computer Programming 89, pp. 71–87

Cabot J., Kolovos D. S. (2016) Human Factors in the Adoption of Model-Driven Engineering: An Educator's Perspective. In: Advances in Conceptual Modeling. Springer, pp. 207–217

Chen C., Song I.-Y., Zhu W. (2007) Trends in conceptual modeling: Citation analysis of the ER conference papers (1979-2005). In: Proceedings of the 11th International Conference on the International Society for Scientometrics and Informatrics, pp. 189–200

Ciccozzi F., Famelis M., Kappel G., Lambers L., Mosser S., Paige R. F., Pierantonio A., Rensink A., Salay R., Taentzer G., Vallecillo A., Wimmer M. (2018) How Do We Teach Modelling and Model-Driven Engineering? A Survey. In: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. MODELS '18. Association for Computing Machinery, pp. 122–129

Cohen L. (1988) Statistical power analysis for the behavioral sciences, 2nd. Edition. Lawrence Earlbaum Associates

Daun M., Brings J., Obe P. A., Pohl K., Moser S., Schumacher H., Rieß M. (2017) Teaching Conceptual Modeling in Online Courses: Coping with the Need for Individual Feedback to Modeling Exercises. In: 2017 IEEE 30th Conference on

Software Engineering Education and Training (CSEE T), pp. 134–143

Eckert C., Cham B., Sun J., Dobbie G. (2016) From design to code: An educational approach. In: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE Vol. 2016-January. cited By 3, pp. 443–448

Embley D. W., Liddle S., Pastor Ó. (2011) Conceptual-Model Programming: A Manifesto In: Handbook of Conceptual Modeling Springer, pp. 3–16

Erdfelder E., Faul F., Buchner A. (1996) GPOWER: A general power analysis program. In: Behavior Research Methods, Instruments, & Computers 28(1), pp. 1–11

Ghiran A.-M., Osman C.-C., Buchmann R. A. (2020) Advancing Conceptual Modeling Education Towards a Generalized Model Value Proposition. In: Advances in Information Systems Development. Springer, pp. 1–18

Hailpern B., Tarr P. (2006) Model-driven development: The good, the bad, and the ugly. In: IBM Systems Journal 45(3), pp. 451–461

Hamou-Lhadj A., Gherbi A., Nandigam J. (2009) The Impact of the Model-Driven Approach to Software Engineering on Software Engineering Education. In: 2009 Sixth International Conference on Information Technology: New Generations, pp. 719–724

Härer F., Fill H.-G. (2020) Past Trends and Future Prospects in Conceptual Modeling - A Bibliometric Analysis. In: Conceptual Modeling. Springer, pp. 34–47

Huelin R., Iheanacho I., Payne K., Sandman K. (2015) What's in a Name? Systematic and Non-Systematic Literature Reviews, and Why the Distinction Matters. In: Evidence Forum. https://www.evidera.com/wp-content/uploads/2015/06/Whats-in-a-Name-Systematic-and-Non-Systematic-Literature-Reviews-and-Why-the-Distinction-Matters.pdf

IEEE (1991) IEEE standard computer dictionary. A compilation of IEEE standard computer glossaries.

Juristo N., Moreno A. (2001) Basics of Software Engineering Experimentation. Springer

Kuzniarz L., Martins L. E. G. (2016) Teaching Model-Driven Software Development: A Pilot Study. In: Proceedings of the 2016 ITiCSE Working Group Reports. ITiCSE '16. Association for Computing Machinery, pp. 45–56

Larenas F., Marın B., Giachetti G. (2018) Classutopia: A Serious Game for Conceptual Modeling Design. In: The 30th International Conference on Software Engineering and Knowledge Engineering, Hotel Pullman, Redwood City, California, USA, July 1-3, 2018. KSI Research Inc. and Knowledge Systems Institute Graduate School, pp. 116–115

Lim D.-J. (2019) Incorporating a Model-Driven Approach into an Embedded Software Course. In: Electronics 8(9)

Martínez Y., Cachero C., Meliá S. (2013) MDD vs. traditional software development: A practitioner's subjective perspective. In: Information and Software Technology 55(2), pp. 189–200

Moody D. L. (2003) The method evaluation model: a theoretical model for validating information systems design methods. In: Proceedings of the 11th European Conference on Information Systems, ECIS 2003, Naples, Italy 16-21 June 2003, pp. 1327–1336

Muller G. (2015) Challenges in Teaching Conceptual Modeling for Systems Architecting. In: Advances in Conceptual Modeling. Springer, pp. 317–326

OMG (2021) Interaction Flow Modeling Language (IFML) : http://www.ifml.org/

Paja E., Horkoff J., Mylopoulos J. (2015) The Importance of Teaching Systematic Analysis for Conceptual Models: An Experience Report. In: Advances in Conceptual Modeling. Springer, pp. 347–357

Panach I., Pastor O. (2021a) Literature review dataset www.uv.es/~joigpana/research/PapersDatasetEMISAJ2021.xlsx

Panach I., Pastor O. (2021b) Zenodo Repository with Experiment Material

Pastor O., Molina J. C. (2007) Model-driven architecture in practice - a software production environment based on conceptual modeling. Springer

Perrenet J. C., Bouhuijs P. A. J., Smits J. G. M. M. (2000) The Suitability of Problem-based Learning for Engineering Education: Theory and practice. In: Teaching in Higher Education 5(3), pp. 345–358

Porubän J., Bačíková M., Chodarev S., Nosál M. (2015) Teaching pragmatic model-driven software development. In: Computer Science and Information Systems 12(2) cited By 4, pp. 683–705

Reyes L. P. Á., Quintero B. C. (Mar. 2020) Teaching based on models and transformations under the active learning approach. In: Journal of Physics: Conference Series 1513, p. 012012

Ringert J. O., Rumpe B., Schulze C., Wortmann A. (2017) Teaching Agile Model-Driven Engineering for Cyber-Physical Systems. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track. ICSE-SEET ’17. IEEE Press, pp. 127–136

Rosenthal K., Ternes B., Strecker S. (2019) Learning Conceptual Modeling: Structuring Overview, Research Themes and Paths for Future Research. In: 27th European Conference on Information Systems - Information Systems for a Sharing Society, ECIS 2019, Stockholm and Uppsala, Sweden, June 8–14, 2019

Roungas B., Dalpiaz F. (2016) A Model-Driven Framework for Educational Game Design. In: Games and Learning Alliance. Springer, pp. 1–11

Selic B. (2003) The Pragmatics of Model-Driven Development. In: IEEE software 20(5), pp. 19–25

Singh Y., Sood M. (2009) Model Driven Architecture: A Perspective. In: Advance Computing Conference, 2009. IACC 2009. IEEE International, pp. 1644–1652

Sitaraman M., Long T. J., Weide B. W., Harner E. J., Wang L. (2002) Teaching Component-Based Software Engineering: A Formal Approach and Its Evaluation. In: Computer Science Education 12(1–2), pp. 11–36

Watson R. T., Webster J. (2020) Analysing the past to prepare for the future: Writing a literature review a roadmap for release 2.0. In: Journal of Decision Systems 29(3), pp. 129–147

WebRatio (2021) WebRatio https://www.webratio.com

West B. T., Welch K. B., Galecki A. T. (2014) Linear mixed models: a practical guide using statistical software. CRC Press

Wohlin C., Runeson P., Höst M., Ohlsson M. C., Regnell B., Wesslén A. (2012) Experimentation in Software Engineering: An Introduction. Springer

Zribi S., Calabrò A., Lonetti F., Marchetti E., Jorquera T., Lorré J. (2016) Design of a simulation framework for model-based learning. In: 2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 631–639